Chapter 1
INTRODUCTION

On June 21, 2000, I was employed as Executive Vice President at Genomica, a bio-informatics company in Boulder, Colorado. I remember the date because my son Asher was born at one o'clock that morning.

His birth was a good start to the day. Asher was actually born on his predicted due date (in the United States this happens about 5% of the time). So we (really my wife, Jenine) had finished our nine-month "project" on schedule. And to top things off, Asher had a very high Apgar score, indicating that we had produced a healthy, good-quality result! Our biggest stakeholder, our older son, Jonah, was thrilled to have a younger brother. On time, high quality, and delighted stakeholders—it truly was a good day!

After a brief nap, I checked email and saw that the CEO of Genomica had sent an urgent message asking me to be at a board of directors' meeting at 8:00 a.m. that same day. Begrudgingly, I left the hospital and went to the meeting.

When I arrived, I was told that the VP of Engineering had been fired the night before and I had now inherited the 90-person engineering team. I wasn't surprised. For several months the executive team and board had been discussing Genomica's inability to deliver valuable products on time and with acceptable quality, and the VP of Engineering was at the center of that discussion.

It was now my responsibility to oversee the effort of substantially improving the results of our product development organization. I remember being struck by the irony of that day's successful delivery and my new responsibilities.

Because I was already quite busy overseeing sales and marketing, I was told that at my discretion I could hire a new VP of Engineering to report to me. The person I chose to hire was Mike Cohn (Cohn 2004; Cohn 2006; Cohn 2010), and Scrum was the approach that we decided to use.

What Is Scrum?

Scrum is an **agile** approach for developing innovative products and services. Figure 1.1 shows a simple, generic, agile development approach.

With an agile approach, you begin by creating a **product backlog**—a prioritized list of the features and other capabilities needed to develop a successful product. Guided by the product backlog, you always work on the most important or highest-priority items first. When you run out of resources (such as time), any work that didn't get completed will be of lower priority than the completed work.

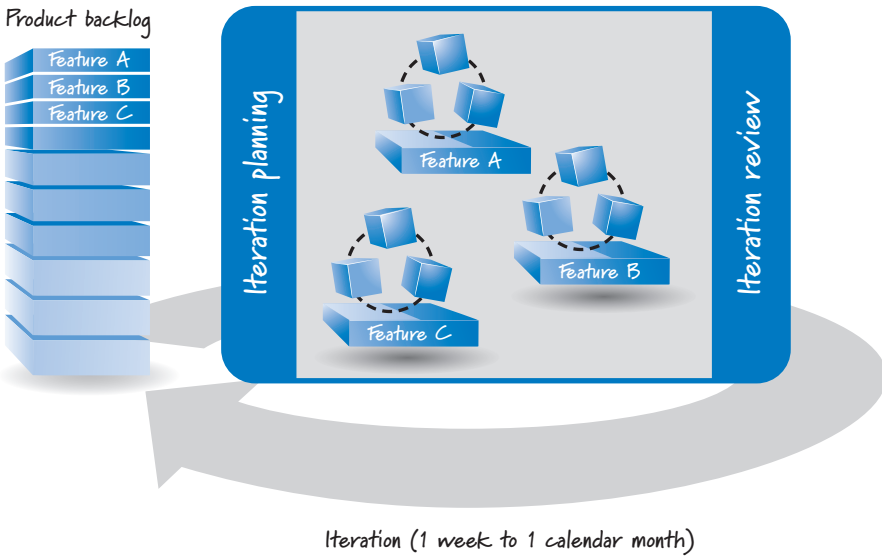

FIGURE 1.1 Agile development overview

The work itself is performed in short, timeboxed **iterations**, which usually range from a week to a calendar month in length. During each iteration, a self-organizing, **cross-functional team** does all of the work—such as designing, building, and testing—required to produce completed, working features that could be put into production.

Typically the amount of work in the product backlog is much greater than can be completed by a team in one short-duration iteration. So, at the start of each iteration, the team plans which high-priority subset of the product backlog to create in the upcoming iteration. In Figure 1.1, for example, the team has agreed that it can create features A, B, and C.

At the end of the iteration, the team reviews the completed features with the stakeholders to get their feedback. Based on the feedback, the product owner and team can alter both what they plan to work on next and how the team plans to do the work. For example, if the stakeholders see a completed feature and then realize that another feature that they never considered must also be included in the product, the product owner can simply create a new item representing that feature and insert it into the product backlog in the correct order to be worked on in a future iteration.

At the end of each iteration, the team should have a potentially shippable product (or increment of the product), one that can be released if appropriate. If releasing after each iteration isn't appropriate, a set of features from multiple iterations can be released together.

As each iteration ends, the whole process is begun anew with the planning of the next iteration.

Scrum Origins

Scrum's rich history can be traced back to a 1986 *Harvard Business Review* article, "The New New Product Development Game" (Takeuchi and Nonaka 1986). This article describes how companies such as Honda, Canon, and Fuji-Xerox produced world-class results using a scalable, team-based approach to **all-at-once product development**. It also emphasizes the importance of empowered, self-organizing teams and outlines management's role in the development process.

The 1986 article was influential in weaving together many of the concepts that gave rise to what today we call Scrum. Scrum is not an acronym, but rather a term borrowed from the sport of rugby, where it refers to a way of restarting a game after an accidental infringement or when the ball has gone out of play. Even if you are not a rugby aficionado, you have probably seen a scrum where the two sets of forwards mass together around the ball with locked arms and, with their heads down, struggle to gain possession of the ball.

Takeuchi and Nonaka used the metaphors of rugby and the scrum to describe product development:

> *The . . . "relay race" approach to product development . . . may conflict with the goals of maximum speed and flexibility. Instead a holistic or "rugby" approach—where a team tries to go the distance as a unit, passing the ball back and forth—may better serve today's competitive requirements.*

In 1993, Jeff Sutherland and his team at Easel Corporation created the Scrum process for use on a software development effort by combining concepts from the 1986 article with concepts from object-oriented development, empirical process control, iterative and incremental development, software process and productivity research, and complex adaptive systems. In 1995, Ken Schwaber published the first paper on Scrum at OOPSLA 1995 (Schwaber 1995). Since then, Schwaber and Sutherland, together and separately, have produced several Scrum-specific publications, including *Agile Software Development with Scrum* (Schwaber and Beedle 2001), *Agile Project Management with Scrum* (Schwaber 2004), and "The Scrum Guide" (Schwaber and Sutherland 2011).

Though Scrum is most commonly used to develop software products, the core values and principles of Scrum can and are being used to develop different types of products or to organize the **flow** of various types of work. For example, I have worked with organizations that have successfully used Scrum for organizing and managing the work associated with hardware development, marketing programs, and sales initiatives.

Why Scrum?

So what made an agile approach like Scrum a good choice for Genomica? First, it was clear that Genomica's previous approach to development simply wasn't working. That was the bad news; the good news was that most everyone agreed.

Genomica operated in a complex domain where more was unknown than known. We built products that had never been built before. Our focus was on bleeding-edge, continuously evolving, state-of-the-art, discovery informatics platforms that research scientists would use to help discover the next blockbuster molecule. We needed a way of developing that would allow us to quickly explore new ideas and approaches and learn fast which solutions were viable and which were not. We had a strategic corporate partner to whom we needed to show working results every few weeks or so to get feedback, because our product had to integrate with its core line of DNA sequencers. This need for rapid exploration and feedback did not mesh well with the detailed, up-front planning we had been doing.

We also wanted to avoid big up-front architecture design. A previous attempt to create a next generation of Genomica's core product had seen the organization spend almost one year doing architecture-only work to create a grand, unified bioinformatics platform. When the first real scientist-facing application was put on top of that architecture, and we finally validated design decisions made many months earlier, it took 42 seconds to tab from one field on the screen to the next field. If you think a typical user is impatient, imagine a molecular biologist with a Ph.D. having to wait 42 seconds! It was a disaster. We needed a different, more balanced approach to design, which included some design up front combined with a healthy dose of emergent, just-in-time design.

We also wanted our teams to be more cross-functional. Historically Genomica operated like most organizations. Development would hand off work to the test teams only after it was fully completed. We now had a desire for all team members to synchronize frequently—daily was the goal. In the past, errors were compounded because important issues were being discussed too late in the development effort. People in different areas weren't communicating frequently enough.

For these reasons, and others, we determined that Scrum would be a good fit for Genomica.

Genomica Results

When we chose to embrace Scrum, it was not well known; the first Scrum book didn't appear until the following year (Schwaber and Beedle 2001). However, we pulled together the available information and did the best we could, which was substantially better than we had done before (see Table 1.1).

From an effort perspective, with Scrum development we required one-tenth the amount of effort (calculated in person-months) compared to our previous use of a

TABLE 1.1 Genomica Scrum Results

Measure	Waterfall	Scrum
Effort	10x	1x
Velocity	1x	7x
Customer satisfaction	Poor	Excellent

plan-driven, **waterfall-style** approach to develop a comparable amount of product functionality. Equally important, the Scrum development progressed at seven times the velocity of the waterfall development, meaning that per unit of time, the Scrum development produced about seven times more valuable features than the waterfall development. Even more compelling was that we delivered the software to our partner in a time frame that met the expectations for the launch of its new hardware platform. This enabled us to reinforce a long-term partnership that substantially increased the shareholder value of Genomica.

Can Scrum Help You?

The Genomica pre-Scrum experience of building features that nobody wanted and delivering those features late and with poor quality is not uncommon. Genomica, like many other organizations, had survived by being no worse than its competitors. I saw the same problems when I first started working in commercial software development in the mid-1980s. And for many, after nearly 30 years, the situation hasn't improved.

Today, if you gathered together your business people and developers and asked them, "Are you happy with the results of our software development efforts?" or "Do you think we deliver good customer value in a timely, economical, and quality manner?" what would they say?

More often than not, the people I meet during my worldwide training and coaching answer both questions with a resounding "No." This is followed by a chorus of "Project failure rate is unacceptably high"; "Deliverables are late"; "Return on investment frequently falls short of expectations"; "Software quality is poor"; "Productivity is embarrassing"; "No one is accountable for outcomes"; "Employee morale is low"; "Employee turnover is too high." Then there's the under-the-breath snicker that accompanies the tongue-in-cheek "There must be a better way."

Yet even with all this discontent, most people seem resigned to the fact that dissatisfaction is just part of the reality of software development. It doesn't have to be.

Organizations that have diligently applied Scrum are experiencing a different reality (see Figure 1.2).

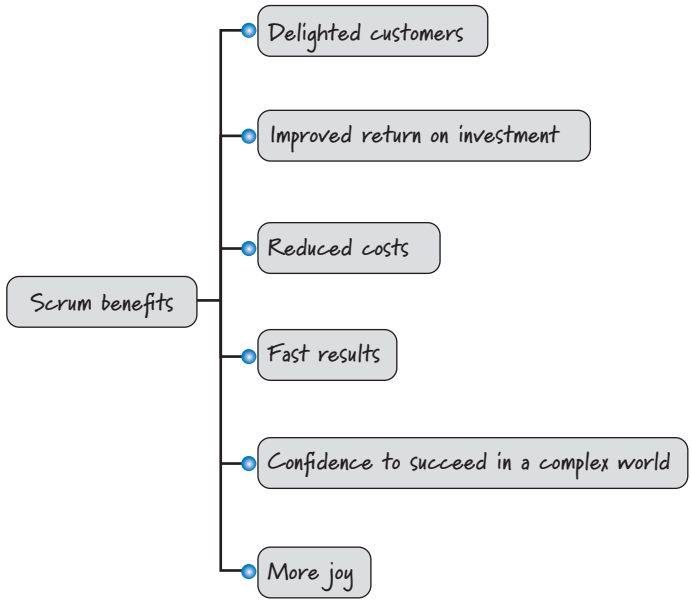

FIGURE 1.2 Scrum benefits

These organizations are repeatedly delighting their customers by giving them what they really want, not just the features they might have specified on the first day when they knew the least about their true needs. They are also seeing an improved return on investment by delivering smaller, more frequent **releases**. And, by relentlessly exposing organizational dysfunction and **waste**, these organizations are able to reduce costs.

Scrum's focus on delivering working, integrated, tested, business-valuable features each iteration leads to results being delivered fast. Scrum is also well suited to help organizations succeed in a complex world where they must quickly adapt based on the interconnected actions of competitors, customers, users, regulatory bodies, and other stakeholders. And Scrum provides more joy for all participants. Not only are customers delighted, but also the people doing the work actually enjoy it! They enjoy frequent and meaningful collaboration, leading to improved interpersonal relationships and greater mutual trust among team members.

Don't get me wrong. Though Scrum is an excellent solution for many situations, it is not the proper solution in all circumstances. The **Cynefin** framework (Snowden and Boone 2007) is a sense-making framework that helps us understand the situation in which we have to operate and decide on a situation-appropriate approach. It defines and compares the characteristics of five different domains: **simple**, **complicated**, **chaotic**, **complex**, and a fifth domain, **disorder**, which occurs when you don't

know which other domain you are in (see Figure 1.3). I will use the Cynefin framework to discuss situations in which Scrum is and is not a good fit.

First, it is important to realize that the many facets of software development and support will not fit nicely into just one Cynefin domain. Software development is a rich endeavor, with aspects that overlap and activities that fall into all of the different domains (Pelrine 2011). So, while most software development work falls in the domains of complicated or complex, to boldly claim that software development is a complex domain would be naive, especially if we define software development to include the spectrum of work ranging from innovative new-product development, ongoing software product maintenance, and operations and support.

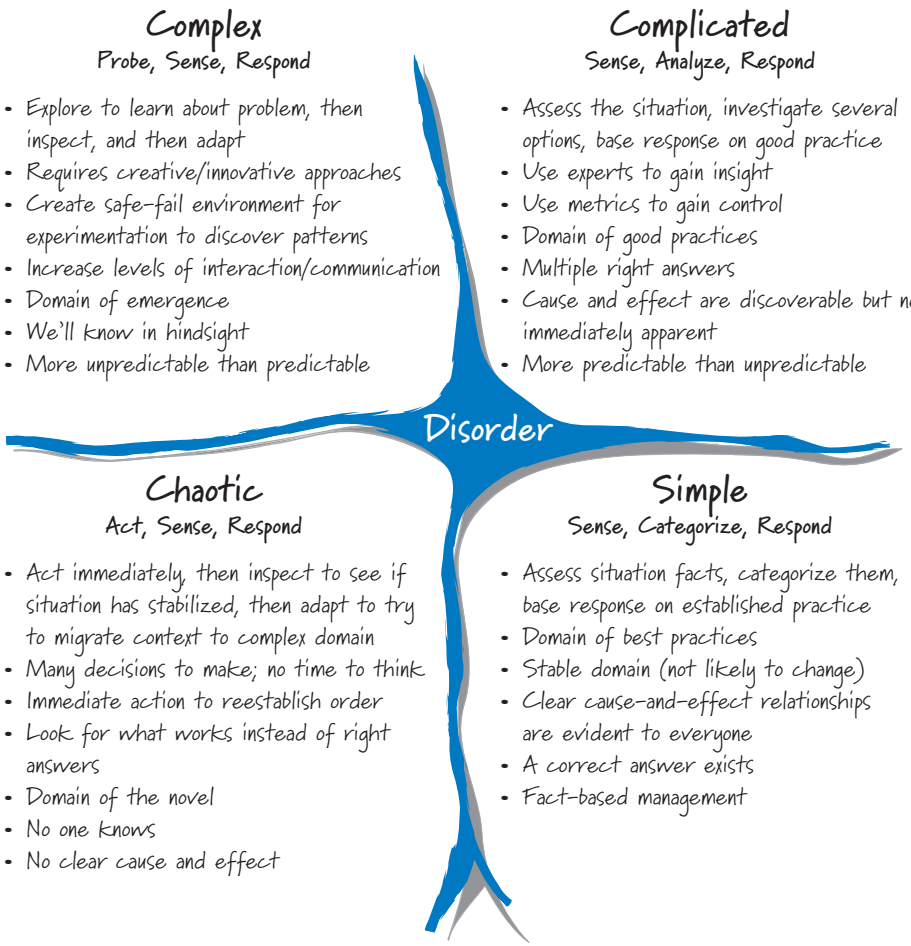

FIGURE 1.3 Cynefin framework

Complex Domain

When dealing with complex problems, things are more unpredictable than they are predictable. If there is a right answer, we will know it only with hindsight. This is the domain of **emergence**. We need to explore to learn about the problem, then **inspect and adapt** based on our learning. Working in complex domains requires creative and innovative approaches. Routine, cookie-cutter solutions simply don't apply. We need to create a safe-fail environment for experimentation so that we can discover important information. In this environment high levels of interaction and communication are essential. Innovative new-product development falls into this category as does enhancing existing products with innovative new features.

Scrum is particularly well suited for operating in a complex domain. In such situations our ability to probe (explore), sense (inspect), and respond (adapt) is critical.

Complicated Domain

Complicated problems are the domain of good practices dominated by experts. There might be multiple right answers, but expert diagnosis is required to figure them out. Although Scrum can certainly work with these problems, it might not be the best solution. For example, a performance optimization effort that calls for adjusting parameters to find the best overall system performance would be better served by assembling experts and letting them assess the situation, investigate several options, and base their response on good practice. Much of day-to-day software maintenance (dealing with a flow of product support or defect issues) falls into this category. This is also where many of the tactical, quantitative approaches like Six Sigma are particularly well suited, although these tactical approaches can also apply with simple domains.

Simple Domain

When dealing with simple problems, everyone can see cause and effect. Often the right answer is obvious and undisputed. This is the domain of legitimate best practices. There are known solutions. Once we assess the facts of our situation, we can determine the proper predefined solution to use. Scrum can be used for simple problems, but it may not be the most efficient tool for this type of problem. Using a process with a well-defined, repeatable set of steps that are known to solve the problem would be a better fit. For example, if we want to reproduce the same product over and over again, a well-defined assembly-line process would be a better fit than Scrum. Or deploying the same commercial-off-the-shelf (COTS) product into the 100th customer environment might best be completed by repeating a well-defined and proven set of steps for installing and configuring the product.

Chaotic Domain

Chaotic problems require a rapid response. We are in a crisis and need to act immediately to prevent further harm and reestablish at least some order. For example, suppose a university published an article stating that our product has a flawed algorithm that is producing erroneous results. Our customers have made substantial business investments based on the results from our product, and they are filing lawsuits against us for large damages. Our lead algorithm designer is on holiday in the jungles of Borneo and can't be reached for two more weeks. Scrum is not the best solution here. We are not interested in prioritizing a backlog of work and determining what work to perform in the next iteration. We need the ability to act immediately and decisively to stem the bleeding. With chaotic problems, someone needs to take charge of the situation and act.

Disorder

You are in the disorder domain when you don't know which of the other domains you are in. This is a dangerous place to be because you don't know how to make sense of your situation. In such cases, people tend to interpret and act according to their personal preference for action. In software development, many people are familiar with and therefore have a personal preference for phase-based, sequential approaches that work well in simple domains. Unfortunately, as I will discuss in Chapter 3, these tend to be a rather poor fit for much of software development. When you are in the disorder domain, the way out is to break down the situation into constituent parts and assign each to one of the other four domains. You are not trying to apply Scrum in the disorder domain; you are trying to get out of this domain.

Interrupt-Driven Work

Scrum is not well suited to highly interrupt-driven work. Say you run a customer support organization and you want to use Scrum to organize and manage your support activities. Your product backlog is populated on a continuous basis as you receive support requests via phone or email. At no point in time do you have a product backlog that extends very far into the future, and the content and order of your backlog could change frequently (perhaps hourly or every few minutes).

In this situation, you will not be able to reliably plan iterations of a week or more because you won't know what the work will be that far into the future. And, even if you think you know the work, there is a very good likelihood that a high-priority support request will arrive and preempt any such forward-looking plans.

In interrupt-driven environments you would be better off considering an alternative agile approach called **Kanban**. Kanban is not a stand-alone process solution, but instead an approach that is overlaid on an existing process. In particular, Kanban advocates that you

- Visualize how the work flows through the system (for example, the steps that the support organization takes to resolve a support request)
- Limit the work in process (WIP) at each step to ensure that you are not doing more work than you have the capacity to do
- Measure and optimize the flow of the work through the system to make continuous improvements

The sweet spots for Kanban are the software maintenance and support areas. Some Kanban practitioners point out that Kanban's focus on eliminating overburden (by aligning WIP with capacity) and reducing variability in flow while encouraging an evolutionary approach to change makes it appropriate to use in complex domains as well.

Scrum and Kanban are both agile approaches to development, and each has strengths and weaknesses that should be considered once you make sense of the domain in which you are operating. In some organizations both Scrum and Kanban can be used to address the different system needs that coexist. For example, Scrum can be used for new-product development and Kanban for interrupt-driven support and maintenance.

Closing

Scrum is not a silver bullet or a magic cure. Scrum can, however, enable you to embrace the changes that accompany all complex **product development efforts**. And it can, and has, worked for Genomica and many other companies that decided to employ an approach to software development that better matched their circumstances.

Although the Scrum framework is simple, it would be a mistake to assume that Scrum is easy and painless to apply. Scrum doesn't prescriptively answer your process questions; instead, it empowers teams to ask and answer their own great questions. Scrum doesn't give individuals a cookbook solution to all of their organizational maladies; instead, Scrum makes visible the dysfunctions and waste that prevent organizations from reaching their true potential.

These realizations can be painful for many organizations. However, if they move past the initial discomfort and work to solve the problems Scrum unearths, organizations can take great strides in terms of both their software development process and products and their levels of employee and customer satisfaction.

The rest of the book is devoted to discussing the essential aspects of Scrum. I will begin with a description of the entire Scrum framework, including its roles, activities, artifacts, and **rules**. Who knows; if you use Scrum in the right way and in the proper conditions, perhaps you too will deliver value as successfully as my wife did on that fateful day back in 2000.